

# Refactoring the Kernel Cryptographic Services Framework

BY JOHN BALDWIN

The FreeBSD kernel includes a cryptographic services framework used by other kernel subsystems for data encryption and authentication. Consumers of this framework include GELI, IPsec, kernel TLS offload, and ZFS. This framework is most commonly referred to by the acronym OCF. Originally, OCF stood for the OpenBSD Cryptographic Framework (<https://www.openbsd.org/papers/crypt-service.pdf>), but over time it has become an acronym for the OpenCrypto Framework.

## History of OCF


The OpenCrypto framework was first imported into FreeBSD 5.0 as a port of the OpenBSD Cryptographic Framework by Sam Leffler in 2002. This initial version was primarily focused on supporting encryption and authentication of network packets for IPsec. It also included a character device driver, `/dev/crypto`, which supported custom `ioctl()` commands to permit userland applications to off-load cryptographic operations to cryptography co-processors.

OCF supported two different modes: symmetric and asymmetric. In symmetric mode, consumers (such as IPsec) first created a session describing parameters such as the algorithms to use and key lengths. Sessions were bound to crypto device drivers (either co-processor drivers or a software driver). Once a session was created, the consumer could issue one or more requests against a session. Each request performed a single operation such as encrypting or decrypting a buffer. Consumers explicitly destroyed a session once it was no longer needed. Asymmetric mode, however, worked differently. Rather than using sessions, each asymmetric operation was dispatched individually, and OCF chose a driver for each operation. Asymmetric operations were intended to assist with public-key cryptography and performed arithmetic on “big-numbers” such as computing a modulus according to the Chinese Remainder Theorem. Asymmetric operations were only used by user processes via `/dev/crypto`.

---

OCF supported two different modes: symmetric and asymmetric.





Symmetric sessions in OCF supported cipher and digest algorithms that were contemporary at the time. Supported ciphers included Cipher Block Chaining (CBC) modes of Data Encryption Standard (DES) and Triple-DES. Digest algorithms included Hash-based Message Authentication Codes (HMAC) constructions of MD5 and SHA-1. In addition, Encrypt-then-Authenticate (EtA) combinations of ciphers and HMACs were also supported.

Over time, a few additional algorithms were added such as SHA-2 digests and AES-XTS. However, the largest set of changes came in FreeBSD 11.0 with the addition of AES-CTR (a stream cipher) and AES-GCM an Authenticated Encryption with Associated Data (AEAD algorithm) by John-Mark Gurney. Modern versions of TLS and IPsec prefer AEAD algorithms and have deprecated other constructions such as EtA.

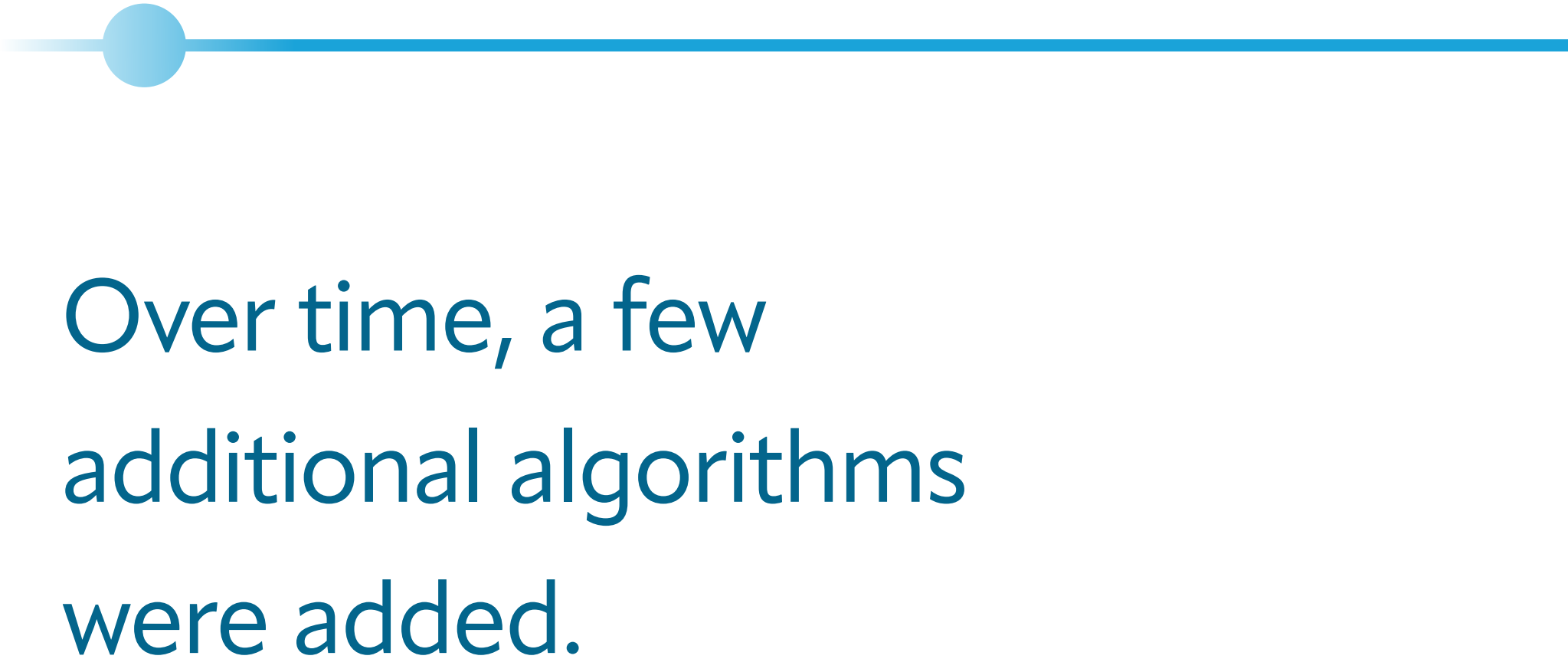
## State of OCF in FreeBSD 11

During the 12.0 development cycle, I ported two Linux crypto drivers to FreeBSD (`ccr(4)` and an out-of-tree driver). This was my first experience with OCF, and I came away feeling that the interface had some quirks that resulted in extra “busy work” in crypto device drivers.

### Linked-Lists for Transforms


Symmetric cryptographic operations in OCF are managed via sessions. OCF consumers create sessions describing the types of operations to be performed (algorithms to use, key sizes, etc.). Individual operations are then invoked on a session permitting drivers to cache state (such as precomputed key schedules) across operations. In 11.0, OCF session parameters were described by a linked-list of structures. Each structure defined either a single data transformation (such as symmetric encryption or compression) or a single digest computation. Sessions using a combination of algorithms (such as EtA) used separate structures for the encryption and authentication steps. Cryptographic operations also used a linked-list of crypto descriptors (one for each session parameter structure) describing the range of the data buffer on which to perform each operation as well as ancillary data such as keys and explicit Initialization Vectors (IVs) or nonces.

OCF in 11.0 did not define a specific order for session structures (e.g., encryption before authentication). Instead, consumers were free to construct the linked-list in an arbitrary order. As a result, drivers first walked the linked-list determining if there were unsupported combinations (e.g. multiple ciphers) as well as saving pointers to supported transforms (typically a cipher pointer and an auth pointer). Once this pass completed, drivers validated transform-specific parameters such as requested algorithms and key lengths. Operation descriptors were similar except that the order mattered. In particular, the `cryptosoft(4)` software driver depended on iterating over each descriptor in sequence to perform the desired set of operations. Other device drivers walked the descriptor chain validating that the chain contained the right number and type of descriptors (matching the session) again saving pointers to specific descriptors (e.g. cipher and auth descriptors) before completing the request.



Over time, a few additional algorithms were added.





Working with the linked-lists was not overly difficult, but it was tedious and resulted in a lot of code duplicated across drivers. Another common theme was that the OCF layer itself did not perform checks that were not driver-specific (such as validating the list of operation descriptors against the session). Instead, every driver was required to duplicate these checks. Similarly, OCF did not perform centralized checks on session parameters such as rejecting requests to create sessions with invalid parameters such as a cipher session with a key length that was not defined for the associated algorithm. These checks were instead duplicated across drivers.

### **AEAD Support**

AEAD algorithms combine both encryption and authentication in a single algorithm. These algorithms provide similar functionality to EtA cipher suites but with a few changes. Notably, AEAD algorithms use a single key and nonce to provide both encryption and authentication. Under the covers, existing AEAD ciphers typically consist of separate encryption and authentication algorithms and derive separate keys and nonces for each from the user-supplied values. For example, AES-GCM uses AES-CTR for its cipher. AES-GCM is commonly used with a 12-byte nonce which is used as the upper 12 bytes of the IV with AES-CTR. The low 4 bytes are used as a counter which is incremented for each block of ciphertext. The keystream for block 1 is used as part of the computation of the authentication tag and the ciphertext is encrypted starting with the keystream for block 2.

In 11.0, OCF used separate session parameters and crypto descriptors for the cipher and authentication "sides". Consumers had to specify the same inputs such as keys and nonces for both sides, and drivers were required to check that both sides were consistent. In addition, due to a quirk of how some parts of OCF managed authentication algorithms, separate algorithm constants were defined for each key size (128, 192, and 256 bits) for the GMAC side of AES-GCM. Again, consumers had to ensure the GMAC constant matched the key size and drivers had to check for mismatches.

Crypto operations for AEAD in 11.0 also worked a bit differently than EtA requiring separate AEAD vs EtA logic in drivers. For EtA, OCF assumed that authentication was performed over a single region containing both auth-only data (such as the ESP header for IPsec) and the ciphertext. Furthermore, it assumed that these regions of the buffer were contiguous in the input data buffer such that they could be described by a single descriptor. This approach worked well with cryptosoft(4)'s implementation. AEAD algorithms, however, require more explicit separation of auth-only data (also known as Additional Associated Data (AAD)) from the ciphertext. AEAD algorithms define the order in which AAD is input into the authentication computation relative to the ciphertext regardless of the location within the associated data buffer. Existing AEAD algorithms also include the length of the AAD and ciphertext

---

**Working with the linked-lists was not overly difficult, but it was tedious and resulted in a lot of code duplicated across drivers.**



regions as inputs into the authentication function. To simplify the implementation of AEAD algorithms, the authentication descriptor for AEAD operations only covered the AAD region. It was implicit that the authentication algorithm must also be executed on the ciphertext region described by the cipher descriptor. Secondly, decryption operations for AEAD algorithms verified the supplied authentication digest, or tag, and failed the request with an error (**EBADMSG**) if it did not match. For EtA, OCF in 11.0 always computed the digest on the input and required the caller to save a copy of the original digest and to compare against the computed digest after the EtA operation completed.

### IV/Nonce Handling

Crypto descriptors in OCF for cipher operations supported a tri-state of possible settings for dealing with IVs or nonces. First, an IV could either be supplied in a location in the data buffer (such as is commonly done in EtA algorithms for IPsec) or in a separate array in the descriptor. This choice was indicated by a flag in the descriptor. In the case that the IV was located in the data buffer, the driver would generate a random IV and insert it into the buffer for encryption operations unless the consumer set a second flag. In practice, none of the drivers in the tree supported generating IVs in hardware, so every driver duplicated the same block of code for managing IVs. This block of code had to check for invalid combinations of flags and, if the second flag wasn't set, call [arc4rand\(9\)](#) to generate a random IV to insert into the data buffer prior to encryption or authentication.

### Session Handles

OCF sessions were named by integer IDs in 11.0. When a driver created a new session, it allocated a driver-private integer ID and returned it to the caller. This integer ID was supplied by the session consumer for each operation associated with a session as well as when removing a session. All existing drivers allocate driver-specific state for each session. When an operation is performed or a session is deleted, drivers use the session's integer ID to locate this driver-specific state. Existing drivers in the tree either used a  $O(n)$  loop to locate the driver-specific state for each request, or they used a lock to protect access to a resizable array of pointers.

### Session Probing

When an OCF consumer creates a new session, the OCF framework chooses a driver to service requests for the new session. In 11.0, this process was simplistic. Drivers registered a list of algorithms supported by OCF during their initialization. When a session was created, the framework would select a driver based on the requested algorithms. However, if the driver failed to create a new session because it did not support one of the other parameters (e.g., a key size, or if a device only supported standalone encryption or authentication but not combined EtA operations), OCF would propagate that failure back to the caller. Specifically, OCF would not try to fallback to another driver. For example, if OCF initially chose

---

**When an OCF consumer creates a new session, the OCF framework chooses a driver to service requests for the new session.**





a coprocessor driver that failed to handle the new session, the framework would not try to use a software driver which could handle the new session instead.

## Streamlining OCF for Drivers and Consumers

As a driver author, OCF felt a bit clunky and required a fair amount of duplicated code among drivers. Some of this duplication was due to flexibility that did not seem needed. For example, using linked-lists for session parameters and operation descriptors permitted an arbitrary number and order of transformations. In practice, however, the operations used either required a single operation at a time, or a combination of one cipher and one authenticator such as EtA. On the other hand, OCF did not include flexibility that would be useful for some use cases. For example, KTLS did not always perform in-place encryption. To cater to OCF in 11 assuming in-place encryption, KTLS support via OCF had to copy the data into the output buffer before handing the data off to a crypto driver. KTLS also does not store its AAD inline in the on-wire format, but instead combines metadata about each TLS record along with some on-wire data from the TLS record to form the AAD.

To make OCF easier to work with, OCF has been refactored in the past couple of years. The primary goal of this refactoring has been to improve ease of use for driver authors. Improving performance is a secondary goal, and it is hoped that reducing complexity will have that benefit by leading to simpler drivers. All the changes described below shipped in FreeBSD 13.0, though the first change was made in 12.0.

### Opaque Session Handles

The first refactoring was implemented in FreeBSD 12.0 by Conrad Meyer. Conrad replaced the integer session IDs used as a handle for OCF sessions with a new `crypto_session_t` opaque type. Under the hood these handles hold a pointer to a per-session structure allocated by the OCF framework. This per-session structure contains memory reserved for driver-specific session state. Drivers now provide the desired size for their driver-specific session state when registering with OCF. Before OCF asks a driver to initialize a new session, OCF allocates memory for the driver-specific session state. Drivers can obtain a pointer to this per-session state at any time via `crypto_driver_session()` as a cheap,  $O(1)$  operation. When a session is freed, OCF zeroes and frees this driver-private structure as well. This removes the need for drivers to manage the lifecycle of driver-specific session structures using a model similar to that of `device_get_softc()` in new-bus.

### Session Parameters

FreeBSD 13.0 introduced a new flat structure to describe symmetric cryptography sessions. This structure (`struct crypto_session_params` documented in [crypto\\_session\(9\)](#)) replaced the linked list of session structures and includes parameters such as key,

---

As a driver author,  
OCF felt a bit clunky  
and required a fair amount  
of duplicated code among  
drivers.



digest, and nonce lengths, session-wide keys, and a mode. Supported modes include stand-alone compression, encryption, and authentication as well as EtA and AEAD modes that combine both encryption and authentication. For device drivers the loop iterating over the linked list of session structures checking for multiple ciphers as well as identifying the cipher vs authentication structures has been removed. Instead, drivers can use switch statements on the mode and algorithm-specific fields (such as `csp_cipher_algorithm` and `csp_auth_algorithm`) to validate session structures and determine if a session is supported.

The parameter structure also includes room for future expansion. Including an explicit mode permits future combinations such as TLS's Mac-then-Encrypt (MtE) to be implemented if desired. In addition, the parameter structure includes a flags field to request optional features. During the initial conversion of drivers, all drivers were updated to reject sessions with a non-zero flags field. As new feature flags are added, drivers can opt-in to supporting sessions with those features by relaxing the checks on the flags field. If at least one driver supports each new feature that is added, this allows consumers to use new features without requiring changes in all crypto drivers. In practice this means that new optional features must be supported by the `cryptosoft(4)` driver.

### Session Probing

FreeBSD 13.0 also introduced a new crypto driver method, `cryptodev_probesession`. When a new symmetric session is created, OCF invokes this new method on each eligible driver. Drivers can examine all the session parameters including the session mode and key lengths to determine if an individual session is supported. If a driver supports a session, it returns a bidding value from this method that OCF uses to choose the best-suited driver. Bidding values are similar to those used by `DEVICE_PROBE(9)` where a positive value indicates an error and the negative value closest to zero is considered the "best" driver. Unlike `DEVICE_PROBE(9)`, there are no special semantics for a return value of zero. Three bidding values are currently defined for coprocessor drivers, accelerated software drivers (such as `aesni(4)`), and plain software drivers.

Previously OCF did not provide a good way of distinguishing accelerated software drivers from coprocessor and plain software drivers. Prior to 13.0, accelerated software drivers were marked as coprocessor ("hardware") drivers to ensure they were preferred to plain software drivers. However, this also meant that userland requests submitted via `/dev/crypto` were enabled for accelerated software drivers by default. If userland software such as OpenSSL is going to use accelerated software instructions (such as AES-NI on x86), it is more efficient for userland to use those instructions directly rather than paying the additional overhead of system calls to encrypt or decrypt data. Userland requests via `/dev/crypto` only make sense when using a coprocessor (and even for many smaller requests the system call overhead can still outweigh the benefits of offloading operations to a coprocessor). The `cryptodev_probesession` hook provides preference for accelerated software drivers while avoiding conflating them with coprocessor drivers.

---

The parameter structure also includes room for future expansion.



## Crypto Requests

FreeBSD 13.0 features a flattened crypto request structure (`struct cryptop` described in [crypto\\_request\(9\)](#)). This structure existed in older FreeBSD versions, but it no longer contains a pointer to a linked-list of descriptors. Instead, the information previously stored in descriptors such as the size and layout of regions in the data buffer such as AAD and payload are now described by members of the structure. Pointers to per-operation keys and separate IVs are stored directly in the structure as well. The new members in the structure assume that symmetric requests operate on a buffer containing AAD, IV, payload, and MAC regions. (Note that some regions are optional depending on the session parameters and request flags.) EtA modes are expected to apply authentication on both the AAD and encrypted payload regions while AEAD modes treat the AAD and payload regions as defined by the associated algorithm.

IV/nonce handling for requests has also been simplified. The OCF layer now generates any random nonces requested by a consumer before passing a request down to drivers. Drivers now only have to determine if the IV is stored inline in the data buffer or as a separate input in the request structure. A new helper function, `crypto_read_iv()`, permits drivers to fetch the IV from a request into a local buffer. This function eliminated duplicated code to read the IV from a request in almost all drivers.

IV/nonce handling for requests has also been simplified.

## Crypto Buffers

FreeBSD 13.0 added additional abstractions for buffers holding data used as inputs and outputs of symmetric cryptographic requests. Prior to 13.0, crypto requests supported different types of data buffers including flat kernel buffers and `struct mbuf` chains. The type of buffer was encoded via flags in the `crp_flags` field of `struct cryptop` and an overloaded pointer pointed to the backing store. Two helper routines for moving data in and out of a crypto request's data buffer (`crypto_copydata()` and `crypto_copyback()`) accepted the flags field and overloaded pointer as arguments to support different data buffer types.

13.0 adds a new `struct crypto_buffer` type to describe a crypto data buffer. The structure includes an enum member which defines the type of the buffer as well as a union of type-specific fields. This permits buffer types which require more than a single pointer to describe. Using a dedicated type also permitted adding support for separate input and output buffers by storing two structures in `struct cryptop`. The existing `crypto_copydata()` and `crypto_copyback()` routines now accept the crypto request in place of the individual fields.

Two new API extensions further reduce duplicated code in drivers. First, new [bus\\_dma\(9\)](#) functions, `bus_dmamap_load_crp()` and `bus_dmamap_load_crp_buffer()`, permit loading a mapping for a crypto data buffer associated with a crypto request. This is primarily useful for coprocessor drivers which need to construct a DMA scatter/gather list to pass on to the coprocessor. Second, a cursor abstraction, primarily useful for software drivers, allows



drivers to iterate over virtual address ranges of a crypto data buffer. Cursors are bound to a crypto buffer when initialized. Drivers can then iterate over a data buffer either by copying data, which implicitly advances the cursor, or explicitly seeking forward. Logic specific to individual data buffer types is isolated in the implementation of crypto cursors rather than duplicated in software drivers. More details on the crypto cursor API can be found in [crypto buffer\(9\)](#). These extensions permit adding new data buffer types without modifying most existing drivers.

Finally, new helper routines have been added on the consumer side of the OCF API that are used to initialize the data buffer in a crypto request. Each crypto buffer data type has dedicated `crypto_use_*`() and `crypto_use_output_*`() routines that initialize a crypto request's data buffers. For example, `crypto_use_buf()` configures a crypto request to use a flat kernel data buffer as its input buffer. If a consumer does not specify a separate output buffer via one of `crypto_use_output_*`(), then the same data buffer is modified in place as both the input and output buffer.

Finally, new helper routines have been added on the consumer side of the OCF API.

### Semantics Changes

Along with these structural changes, OCF in 13.0 also enforces several semantic changes. Some of these changes fall out from the structural changes while others are intentional towards the goal of simplifying drivers.

1. Sessions can now use at most one cipher and one authentication algorithm.
2. Sessions can only combine multiple algorithms in specific modes. For example, a session cannot mix compression and encryption.
3. Sessions can either use per-operation or per-session keys but not both.
4. Sessions which use per-operation keys instead of per-session keys must use the same key lengths for all operations.
5. AEAD sessions now use a single algorithm constant and key.
6. EtA sessions now validate checksums and fail operations with a bad MAC with `EBADMSG` similar to AEAD sessions.
7. Accelerated software drivers such as `aesni(4)` are now marked as software drivers instead of hardware drivers.

Existing consumers generally required only modest changes. Primarily these consisted of coping with structural changes such as using the session parameters structure. The only change that did not fall into this category was the change to validate MACs for EtA sessions. However, this generally simplified consumers by aligning code paths between AEAD and EtA sessions.

### Driver Testing

The initial import of OCF included limited support for validating crypto drivers. The `tools/tools/crypto` subdirectory contained several utilities. Most of these fetched statistics for specific drivers or subsystems. One utility, `cryptotest.c`, did support some testing,



but it was primarily focused on measuring performance. For encryption algorithms it both encrypted and decrypted a random buffer and verified that the decryption result matched the original plaintext. However, it did not verify if the encrypted message matched a known-good standard. Similarly, for authentication algorithms this tool did no verification at all. It simply measured the performance of performing  $N$  operations.

Along with the changes to support AES-CTR and AES-GCM in 11.0, John-Mark Gurney added support for validating drivers against a set of Known Answer Tests (KAT) published by the [National Institute of Standards and Technology](#). The test vectors can be installed via the [security/nist-kat](#) port or package. The `test/sys/openssl/cryptotest.py` script is able to run these tests against crypto drivers and report any failures.

These two tests did have a few limitations. Both tests only supported a subset of algorithms supported by OCF. The KAT tests were an improvement over `cryptotest.c` since they validated encryption results against a trusted third party. However, the error reporting from the KAT tests was not detailed, and it was not easy to run an individual test against a driver when investigating a mismatch rather than the full battery of tests.

13.0 adds a new testing utility: `tools/tools/crypto/cryptocheck.c`. This utility uses OpenSSL's software cryptography as a gold standard to compare driver output against. This permits testing a broader range of algorithms. The `cryptocheck` utility also permits testing either individual operations or a set of operations spanning different sizes and/or algorithms. While the parameters such as keys and data buffers are populated with random data for each test, the userland RNG is not seeded so that the specific data inputs for individual tests are repeatable across multiple runs. Various parameters can be specified for tests including the sizes of plaintext buffers, keys, AAD, nonces, and MACs. For EtA and AEAD algorithms, `cryptocheck` also verifies that corrupted encryption buffers are detected and rejected with an error.

### Documentation

The existing [crypto\(9\)](#) manual page has been updated and split into several pages. `crypto_session(9)` describes the session parameter structure and APIs to create and manage symmetric sessions. `crypto_request(9)` describes the symmetric crypto request structure and related APIs. `crypto_buffer(9)` describes crypto buffer cursors and other APIs that work on crypto request data buffers. [crypto\\_driver\(9\)](#) describes APIs for use by crypto drivers that are not described in one of the other pages. The [crypto\(7\)](#) page has been reformatted as a list of tables grouped by algorithm type and extended to cover all of the algorithms supported by OCF.

---

Various parameters can be specified for tests including the sizes of plaintext buffers, keys, AAD, nonces, and MACs.



## Subsequent Changes

The set of changes above in 13.0 were authored by myself and mostly landed as a single [commit](#). Since then, OCF has been further extended by various developers.

Alan Somers [added](#) a new type of crypto data buffer that contains a list of VM pages. This permitted the use of unmapped I/O with GELI which improved performance by eliminating page table and TLB maintenance operations. Due to the abstractions around crypto data buffers, Alan's changes only touched a small number of crypto drivers directly. Most drivers worked with the new buffer type without requiring any changes.

I added support for [separate output buffers](#) and [separate AAD buffers](#) as new session feature flags. These improve the performance for kernel TLS by removing the need for data copies and for allocating temporary I/O vectors (struct iovec arrays). Since these requests were added as optional features, only drivers which wished to support kernel TLS needed to be updated to support these features.

Marcin Wojtas from Semihalf added another session feature to support [extended sequence numbers](#) (ESN) in IPsec for non-AEAD ciphers.

Support for additional AEAD ciphers have also been added. AES-CCM (used by OpenZFS) was included in 13.0. ChaCha20-Poly1305 (used by TLS and WireGuard) shipped in 13.1.

13.0 also removes support for older, deprecated ciphers and authenticators such as DES, TripleDES, Blowfish, and MD5-HMAC.

## Conclusion

OCF still has lots of room for improvement, but the refactoring in 13.0 has succeeded in streamlining the API reducing code duplication and "busy" work in both drivers and consumers. (Mark Johnston told me that two OCF drivers he added in 13.0 were much easier to write due to the refactoring.) The changes sufficiently improved performance to permit kernel TLS to switch to using OCF instead of a private software crypto interface. The refactoring also provided a flexible base upon which other developers have been able to extend. I wish to thank Chelsio Communications and Netflix for sponsoring my OCF work in 13.0.

---

**JOHN BALDWIN** is a systems software developer. He has directly committed changes to the FreeBSD operating system for over twenty years across various parts of the kernel (including x86 platform support, SMP, various device drivers, and the virtual memory subsystem) and userspace programs. In addition to writing code, John has served on the FreeBSD core and release engineering teams. He has also contributed to the GDB debugger. John lives in Concord, California with his wife, Kimberly, and three children: Janelle, Evan, and Bella.

OCF still has lots of room for improvement, but the refactoring in 13.0 has succeeded in streamlining the API.